

LAPORAN TUGAS BESAR

IF3270 PEMBELAJARAN MESIN

Implementasi Feed Forward Neural Network



Disusun oleh:

Ibrahim Ihsan Rasyid	13522018
Panji Sri Kuncara Wisma	13522028
Muhammad Syarafi Akmal	13522076

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

2025

DAFTAR ISI

DAFTAR ISI	2
BAB I	
DESKRIPSI PERSOALAN	3
BAB II	
PEMBAHASAN	5
PENJELASAN IMPLEMENTASI	5
A. Deskripsi Kelas	5
B. Forward Propagation	7
C. Backward Propagation dan Weight Update	8
HASIL PENGUJIAN	10
A. Pengaruh Depth dan Width	10
Banyak Layer (Depth)	10
Banyak Neuron tiap Layer (Width)	11
B. Pengaruh Fungsi Aktivasi	11
C. Pengaruh Learning Rate	11
Fungsi Aktivasi eLU	12
Fungsi Aktivasi Hyperbolic Tan	13
Fungsi Aktivasi Sigmoid	13
Fungsi Aktivasi reLU	14
Fungsi Aktivasi Linear	14
D. Pengaruh Inisialisasi Bobot	15
E. Perbandingan dengan Library sklearn	15
BAB III	
KESIMPULAN DAN SARAN	16
KESIMPULAN	16
SARAN	16
BAB IV	
PEMBAGIAN TUGAS	17
BAB V	
REFERENSI	18

BAB I

DESKRIPSI PERSOALAN

Neural Network merupakan salah satu model dalam *Machine Learning* yang telah banyak digunakan dalam berbagai bidang, seperti pengenalan pola, klasifikasi, dan regresi. Model ini meniru cara kerja otak manusia dengan menggunakan jaringan neuron buatan untuk melakukan perhitungan yang kompleks. Salah satu jenis *neural network* yang paling mendasar adalah *Feed Forward Neural Network* (FFNN), yang memiliki arsitektur sederhana di mana informasi mengalir dalam satu arah, dari input layer ke output layer, tanpa loop atau umpan balik.

Dalam tugas besar ini, kami mengimplementasikan FFNN dari awal tanpa menggunakan library seperti TensorFlow atau PyTorch. Implementasi ini bertujuan untuk memberikan pemahaman yang lebih dalam mengenai cara kerja internal *neural network*, termasuk *forward propagation*, *backward propagation*, serta pembaruan bobot menggunakan algoritma *gradient descent*.

Tugas ini menuntut implementasi sebuah modul FFNN yang memenuhi spesifikasi berikut:

1. **Fleksibilitas Arsitektur:** Model harus dapat menerima jumlah neuron dari setiap layer serta fungsi aktivasi yang digunakan.
2. **Dukungan Fungsi Aktivasi:** Implementasi harus mendukung berbagai fungsi aktivasi, yaitu Linear, ReLU, Sigmoid, Hyperbolic Tangent (tanh), dan Softmax.
3. **Dukungan Fungsi Loss:** Model harus dapat menggunakan berbagai fungsi loss, seperti Mean Squared Error (MSE), Binary Cross-Entropy, dan Categorical Cross-Entropy.
4. **Inisialisasi Bobot:** Model harus memiliki beberapa metode inisialisasi bobot, termasuk Zero Initialization, Random dengan distribusi uniform, dan Random dengan distribusi normal.
5. **Forward dan Backward Propagation:** Model harus mampu melakukan perhitungan *forward propagation* untuk mendapatkan output serta *backpropagation* untuk menghitung gradien guna memperbarui bobot dengan algoritma *gradient descent*.
6. **Visualisasi dan Analisis:** Model harus mampu menampilkan struktur jaringan dalam bentuk graf, menampilkan distribusi bobot serta gradien, serta melakukan penyimpanan dan pemuatan model.
7. **Pelatihan dan Evaluasi:** Model harus mendukung pelatihan menggunakan *batch*, dengan parameter seperti *batch size*, *learning rate*, dan jumlah epoch. Selain itu, model harus menyimpan histori pelatihan dan memungkinkan analisis terhadap berbagai *hyperparameter*, seperti *depth*, *width*, fungsi aktivasi, *learning rate*, dan metode inisialisasi bobot.
8. **Perbandingan dengan Library:** Model yang telah diimplementasikan akan dibandingkan dengan Multi-Layer Perceptron (MLP) dari Scikit-Learn menggunakan dataset MNIST 784.

Tujuan dari tugas ini antara lain:

- Mengimplementasikan FFNN dari awal untuk memahami cara kerja internal neural network.
- Menganalisis pengaruh berbagai hyperparameter terhadap performa model.
- Membandingkan hasil implementasi FFNN dengan model yang tersedia di library Scikit-Learn.

- Mengembangkan keterampilan dalam melakukan eksperimen dan analisis model Machine Learning.

Dengan adanya tugas ini, besar harapan kami dapat memperoleh pemahaman mendalam mengenai prinsip dasar neural network dan proses optimasi yang terjadi dalam model *Machine Learning*.

BAB II

PEMBAHASAN

PENJELASAN IMPLEMENTASI

A. Deskripsi Kelas

1. Kelas Layer: Layer adalah kelas yang berguna sebagai *instance* dari objek yang merepresentasikan sebuah layer pada *artificial neural network*. Kelas ini mengandung properti dan metode:
 - a. input: nilai yang menjadi input dari parameter bobot ke layer selanjutnya (array of float)
 - b. a_func: fungsi aktivasi (sebuah callback function)
 - c. w_init: metode inisiasi bobot (sebuah callback function)
 - d. net: hasil net dari layer sebelumnya (float)
 - e. output: hasil keluaran dari layer (apabila sebuah output neuron) (array of float)
 - f. error: error term dari setiap neuron pada layer (array of float)
 - g. weight_to: bobot ke layer selanjutnya (array of float)
 - h. weight_grad: perubahan bobot

```
class Layer:
    def __init__(self, a_func, w_init, neurons):
        self.neurons = neurons
        self.a_func = a_func
        self.w_init = w_init
        self.input = None
        self.net = None
        self.output = None
        self.error = None
        self.weight_to = None
        self.weight_grad = None
```

Gambar A.1 Init Kelas Layer

2. Kelas ANN: ANN adalah kelas yang menjadi interface utama terhadap model yang isinya berupa gabungan dari layer-layer yang telah disusun untuk membentuk sebuah *artificial neural network*. Kelas ini mengandung properti dan metode:
 - a. err_func: fungsi loss (sebuah callback function)
 - b. network: gabungan dari kelas Layer (array of Layer)
 - c. history: variabel untuk mengisi history pembelajaran
 - d. reg: jenis regularisasi yang diimplementasikan
 - e. lambda_reg: nilai lambda dalam regularisasi

- f. `forward_propagation()`: metode untuk melakukan forward propagation
- g. `backward_propagation()`: metode untuk melakukan backward propagation
- h. `evaluate()`: mengevaluasi prediksi dengan label asli dan menghitung errornya
- i. `update_weights()`: melakukan pembaruan terhadap bobot menggunakan error term yang dihitung di backward propagation
- j. `train()`: melakukan pelatihan terhadap bobot yang berisi forward propagation, backward propagation, dan weight update (epoch)
- k. `predict()`: melakukan forward propagation terhadap suatu input dataset
- l. `show()`: menunjukkan konfigurasi network dan ditampilkan di CLI
- m. `save()`: menyimpan model dalam bentuk binary file
- n. `load()`: menggunakan input model dalam bentuk binary file
- o. `visualize_network()`: mengeluarkan output visualisasi dari struktur network
- p. `w_dist_show()`: mengeluarkan grafik plotting terhadap distribusi nilai bobot
- q. `wg_dist_show()`: mengeluarkan grafik plotting terhadap distribusi gradien bobot
- r. `plot_training_history()`: mengeluarkan grafik terhadap history pembelajaran model

```
class ANN:
    def __init__(self, data=None, config=None, input=None, output=None, error=None, load_path=None, reg=None, lambda_reg=0.0):
        self.data = data
        self.err_func = error
        self.history = {"train_loss": [], "val_loss": []}
        self.reg = reg
        self.lambda_reg = lambda_reg

        if load_path:
            self.load(load_path)
            return

        if config is None:
            config = []

        self.network = []
        if input:
            self.network.append(input)
        self.network.extend(config)
        if output:
            self.network.append(output)

        for i in range(len(self.network)-1):
            if self.network[i].w_init[0] is not None:
                self.network[i].weight_to = self.network[i].w_init[0](
                    self.network[i].neurons,
                    self.network[i+1].neurons,
                    self.network[i].w_init[1]
                )
            self.network[i].weight_grad = np.zeros_like(self.network[i].weight_to)
```

B. *Forward Propagation*

Forward propagation merupakan perambatan neural network yang menghasilkan output dari input melalui beberapa proses ‘filter’ atau ‘layer’ untuk mencapai output yang diharapkan. Pada tugas besar ini, proses forward propagation menggunakan skema perkalian matriks batch processing yang ada di salindia kuliah pembelajaran mesin. Berikut merupakan perumusan utama:

$$f(\mathbf{X}; \mathbf{W}_{xh}, \mathbf{W}_{hy}) = f_2(f_1(\mathbf{X}\mathbf{W}_{xh}) \cdot \mathbf{W}_{hy})$$

Rumus ini berarti bahwa net dari setiap neuron di sebuah layer adalah $\mathbf{X}\mathbf{W}_{xh}$, di mana:

- X adalah matriks input layer sebelumnya
- \mathbf{W}_{xh} adalah matriks bobot dari layer sebelumnya ke *current* layer
- $f_x()$ adalah fungsi aktivasi

Konfigurasi matriks dari X:

1	$i[1][1]$	$i[1][..]$	$i[1][y]$
1	$i[..][1]$	$i[..][..]$	$i[..][y]$
1	$i[x][1]$	$i[x][..]$	$i[x][y]$

x: data instance ke-x

y: input ke y

1: input nilai untuk bias

Konfigurasi matriks \mathbf{W}_{xh} :

bias[1]	bias[..]	bias[y]
$w[1][1]$	$w[1][..]$	$w[1][y]$
$w[x][1]$	$w[x][..]$	$w[x][y]$

x: bobot dari neuron x

y: menuju neuron y

Proses perambatannya merupakan perkalian bobot dengan input nilai yang didapatkan melalui net dari layer sebelumnya yang telah difilter menggunakan fungsi aktivasi.

Berikut adalah kode implementasi dari Forward Propagation:

```
def forward_propagation(self, X):
    batch_size = X.shape[0]

    X_with_bias = np.hstack((np.ones((batch_size, 1)), X))

    self.network[0].input = X_with_bias

    # Process through each layer
    for i in range(len(self.network)-1):
        self.network[i+1].net = self.network[i].input @ self.network[i].weight_to

        if i == len(self.network)-2:
            if self.network[i+1].a_func is not None:
                self.network[i+1].output = self.network[i+1].a_func(self.network[i+1].net)
            else:
                self.network[i+1].output = self.network[i+1].net
            self.network[i+1].input = self.network[i+1].output
        else:
            self.network[i+1].output = self.network[i+1].a_func(self.network[i+1].net)
            self.network[i+1].input = np.hstack((np.ones((batch_size, 1)), self.network[i+1].output))

    return self.network[-1].output
```

C. *Backward Propagation dan Weight Update*

Backward propagation adalah proses perambatan neural network yang berfungsi untuk mengevaluasi error/galat dari output berdasarkan label/target yang nanti menjadi nilai kontribusi error untuk setiap neuron di setiap layer (kecuali input layer).

Algoritma backward propagation dalam kode ini diimplementasikan dengan beberapa pertimbangan khusus:

1. Penanganan Error di Output Layer

- Kombinasi Sigmoid-MSE/SSE:

Perhitungan dilakukan dengan mengalikan langsung turunan loss (output - target) dengan turunan sigmoid (output * (1-output)).

- Kombinasi Softmax-CCE:

Error dihitung hanya sebagai (output - target) tanpa mengalikan dengan turunan aktivasi.

- Kombinasi Umum

Error term dihitung dengan mengalikan turunan fungsi loss dengan turunan fungsi aktivasi, mengikuti prinsip dasar chain rule.

2. *Weight Update*

Setelah error term dihitung untuk setiap neuron, langkah selanjutnya adalah menghitung *gradient* untuk setiap *weight*:

```
73         # Weight gradients
74         for i in range(len(self.network)-1):
75             self.network[i].weight_grad = self.network[i].input.T @ self.network[i+1].error
76
```

Proses ini melakukan:

- Pengambilan input dari layer saat ini dan mentranspose-nya
- Perkalian matriks dengan *error term* dari layer berikutnya

Pendekatan matrix-based digunakan karena:

- Menghindari loop per-neuron dan per-weight yang lambat
- Memanfaatkan operasi NumPy yang dioptimalkan untuk komputasi paralel
- Menghitung *gradient* untuk semua weight sekaligus dalam satu operasi matriks

Setelah *gradient* dihitung, *weight update* dilakukan dengan:

```
def update_weights(self, learning_rate):
    batch_size = self.network[0].input.shape[0]
    for i in range(len(self.network)-1):
        self.network[i].weight_to -= learning_rate * (self.network[i].weight_grad / batch_size)
```

Proses ini mencakup:

- Normalisasi *gradient* dengan membaginya dengan batch size
- Pengaturan besarnya perubahan dengan mengalikan *gradient* dengan *learning rate*
- Pengurangan nilai *weight* saat ini dengan hasil perhitungan sebelumnya

Normalisasi kami terapkan dengan harapan untuk:

- Menjaga konsistensi besar perubahan *weight* terlepas dari ukuran batch
- Mencegah perubahan *weight* yang terlalu besar dengan batch besar

HASIL PENGUJIAN

Berikut adalah konfigurasi yang kami gunakan pada pengujian yang kami lakukan

Elemen	Nilai
Banyak layer (depth)	3
Banyak neuron tiap layer (width)	32

Fungsi aktivasi (kecuali layer output)	ReLU
Fungsi loss	Categorical Cross-Entropy (CCE)
Learning rate	0.1
Banyak Epoch	10
Inisialisasi bobot	Xavier initialization
Regularisasi	Tidak ada (None)
Lambda Regularisasi	0.01

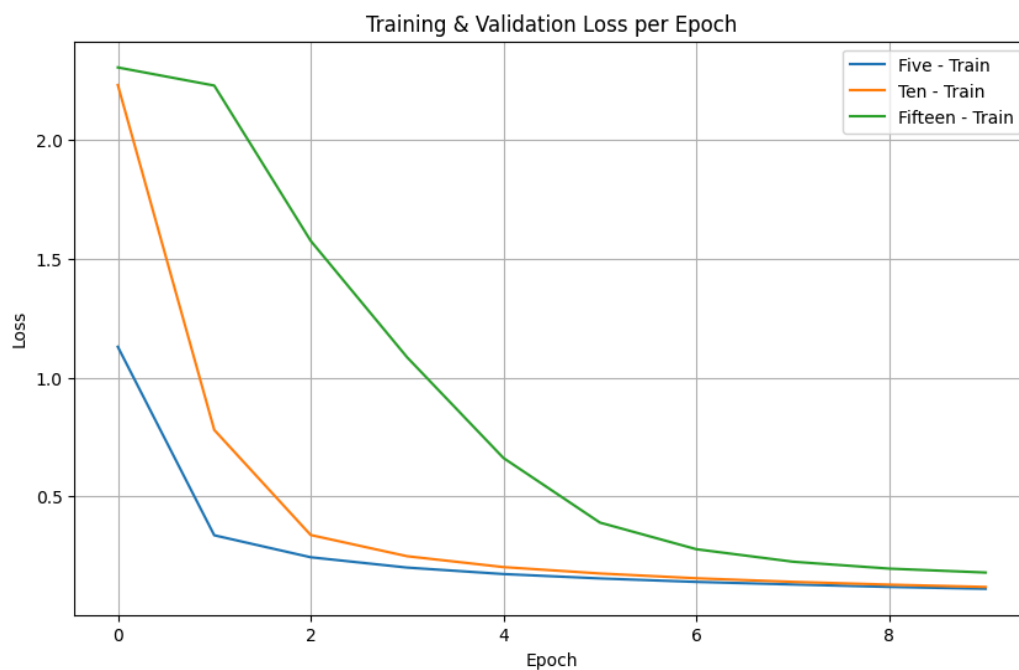
A. Pengaruh Depth dan Width

Banyak Layer (Depth)

Tabel hasil pengujian terhadap banyak layer

Banyak Layer	Akurasi
5	95.31%
10	95.51%
15	92.54%

Grafik training loss dan validation loss tiap epoch

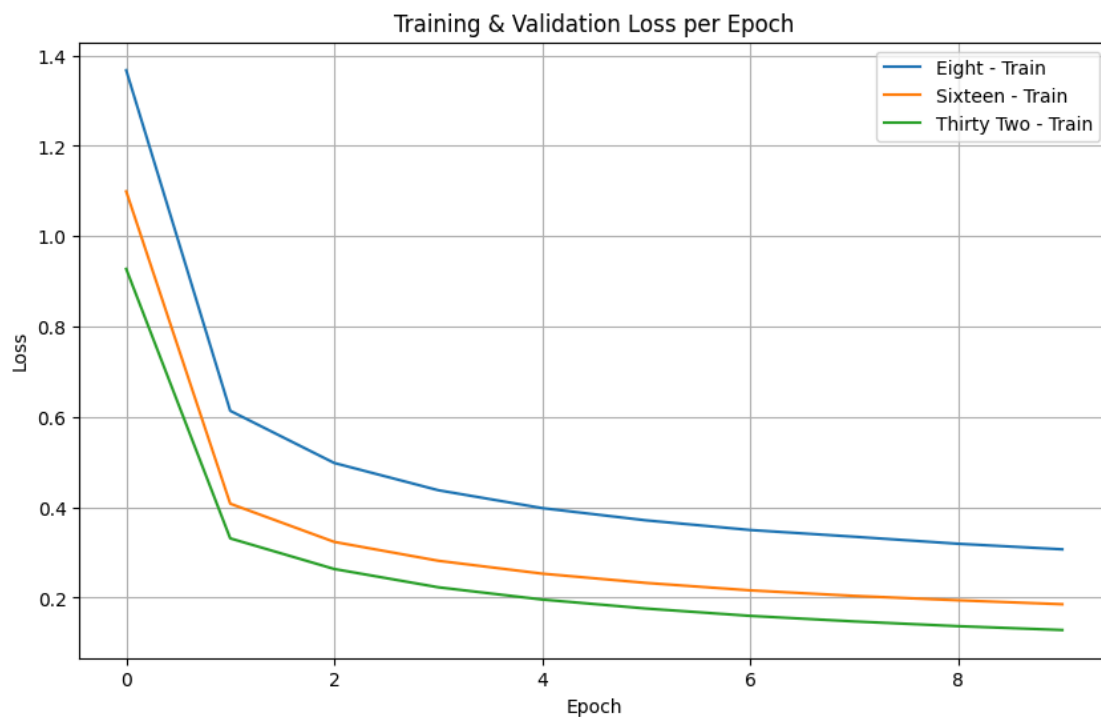


Hasil menunjukkan bahwa kedalaman jaringan memengaruhi kinerja model. Akurasi tertinggi dicapai oleh model dengan 10 layer (95.51%), sedikit lebih baik dari 5 layer (95.31%), sedangkan penambahan hingga 15 layer justru menurunkan akurasi menjadi 92.54%. Grafik menunjukkan model 5 layer konvergen paling cepat, diikuti 10 layer yang stabil, sementara 15 layer mengalami konvergensi lambat. Hal ini mengindikasikan bahwa penambahan layer dapat meningkatkan performa hingga batas tertentu, namun kedalaman berlebih justru menghambat proses pelatihan dan menurunkan akurasi.

Banyak Neuron tiap Layer (Width)

Banyak Neuron tiap Layer	Akurasi
5	85.19%
10	92.77%
20	94.25%

Grafik training loss dan validation loss tiap epoch



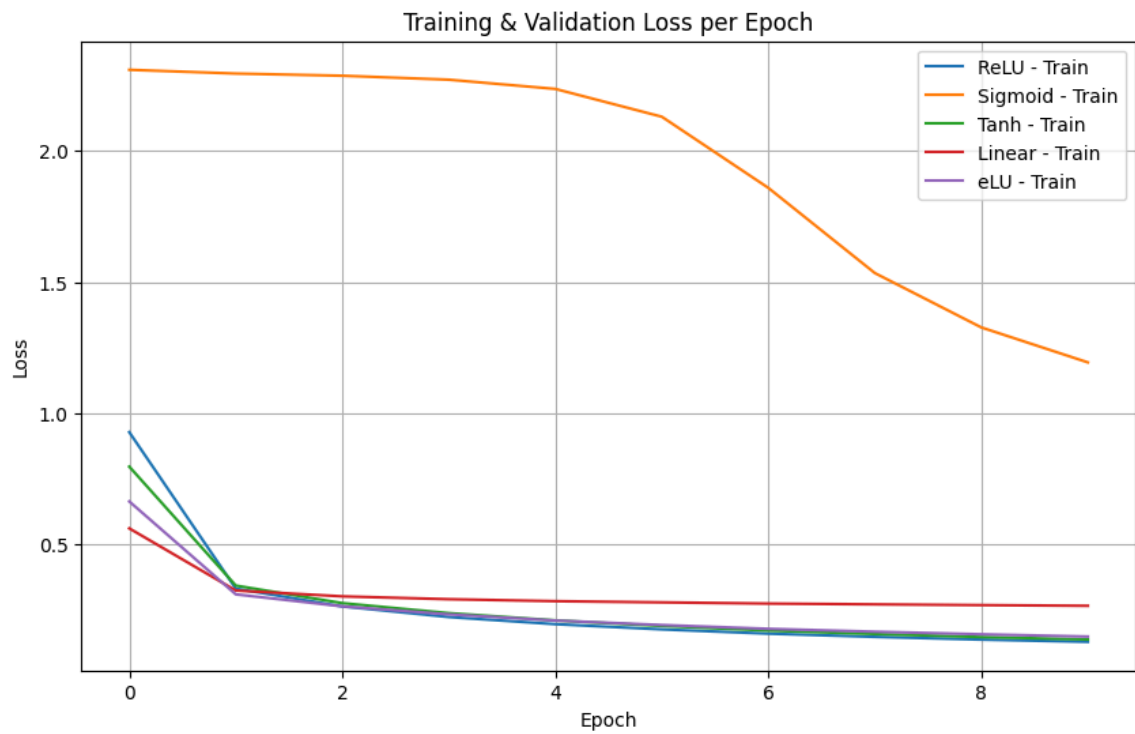
Grafik di atas menunjukkan bahwa nilai width yang terlalu kecil mengurangi akurasi dari model dan peningkatannya mempengaruhi kenaikan akurasi model. Pada rentang [10, 65], terlihat bahwa pertumbuhan grafik mulai landai. Depth yang digunakan pada pengujian ini adalah 1.

B. Pengaruh Fungsi Aktivasi

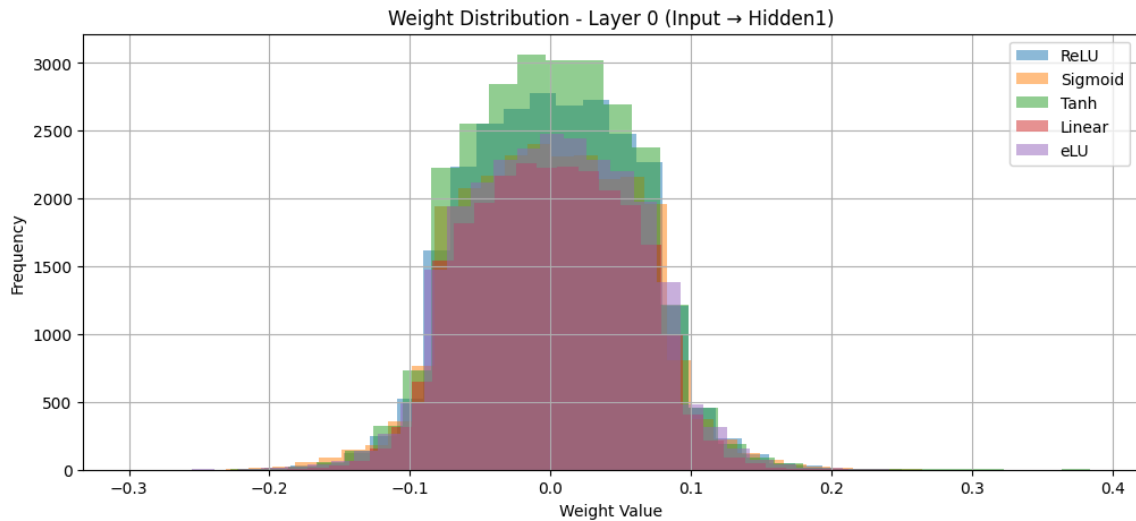
Tabel hasil pengujian terhadap fungsi aktivasi

Fungsi Aktivasi	Akurasi
ReLU	95.44%
Sigmoid	63.48%
Hyperbolic Tan	95.46%
Linear	92.06%
eLU	95.07%

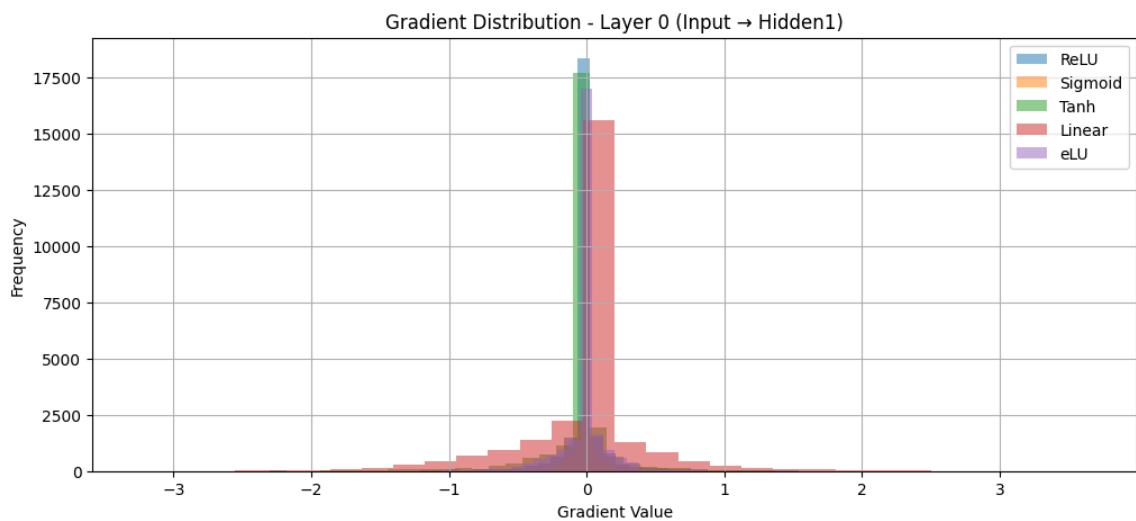
Grafik training loss dan validation loss tiap epoch



Distribusi bobot layer 1



Distribusi gradien bobot layer 1



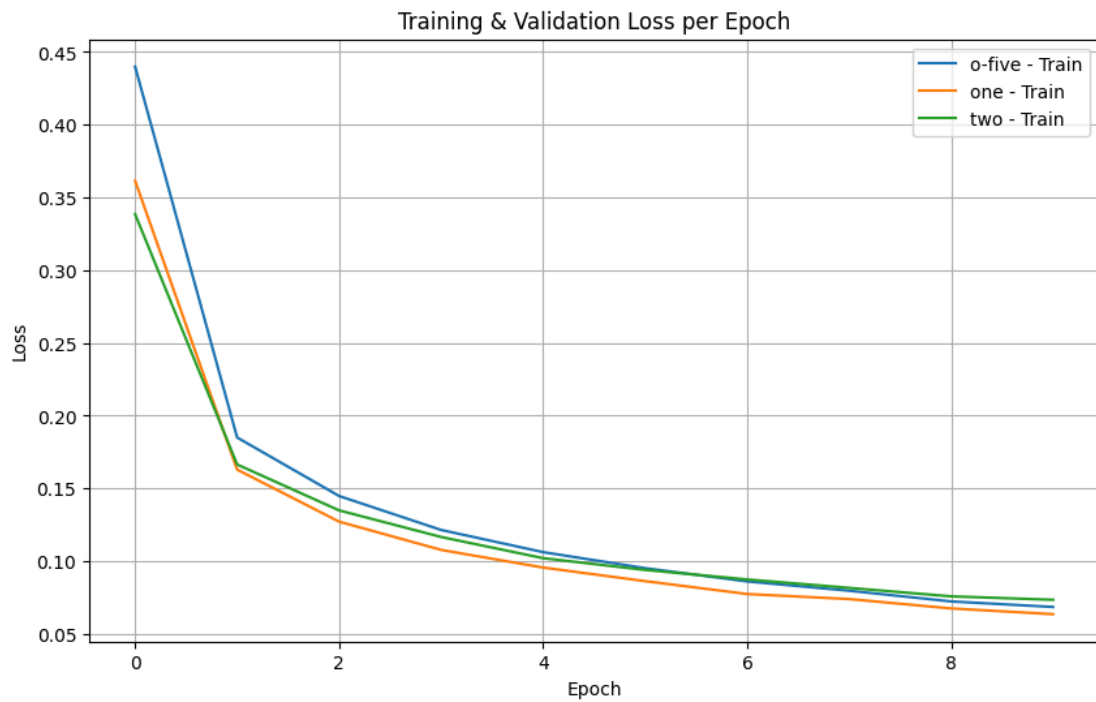
Hasil pengujian menunjukkan bahwa fungsi aktivasi sangat memengaruhi performa model. Tanh dan ReLU memberikan akurasi tertinggi (masing-masing 95.46% dan 95.44%), diikuti oleh eLU (95.07%) dan Linear (92.06%), sementara Sigmoid menghasilkan akurasi terendah (63.48%). Grafik loss memperlihatkan bahwa Sigmoid mengalami kesulitan dalam konvergensi dan stagnan di nilai loss tinggi. Distribusi bobot dan gradien juga menunjukkan bahwa Sigmoid menghasilkan gradien yang sangat kecil di sebagian besar unit. Sebaliknya, fungsi seperti ReLU, Tanh, dan eLU menjaga distribusi gradien lebih seimbang, sehingga mungkin lebih efektif dalam pelatihan jaringan.

C. Pengaruh Learning Rate

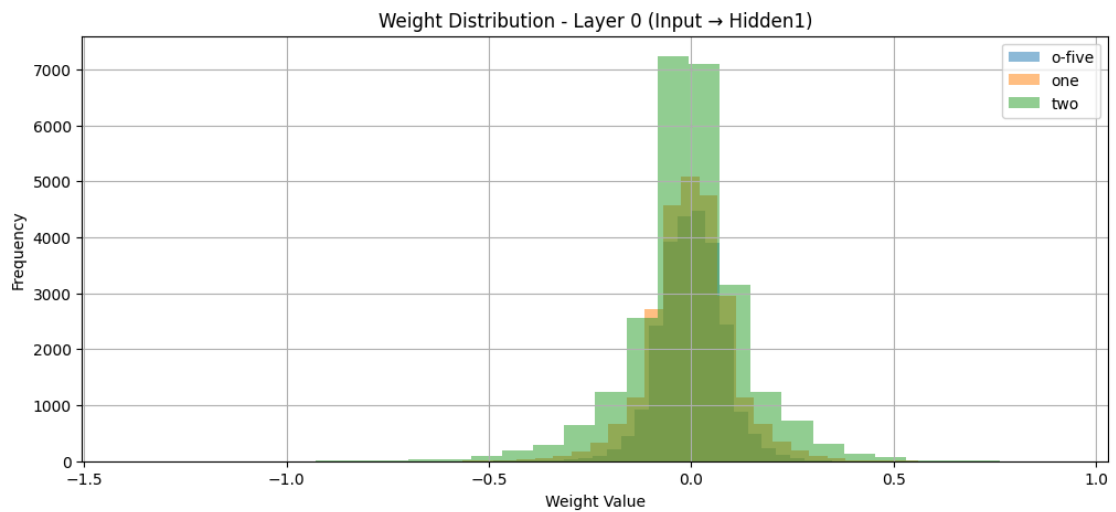
Tabel hasil pengujian terhadap learning rate

Learning Rate	Akurasi
0.05	95.84%
0.1	96.37%
0.2	96.48%

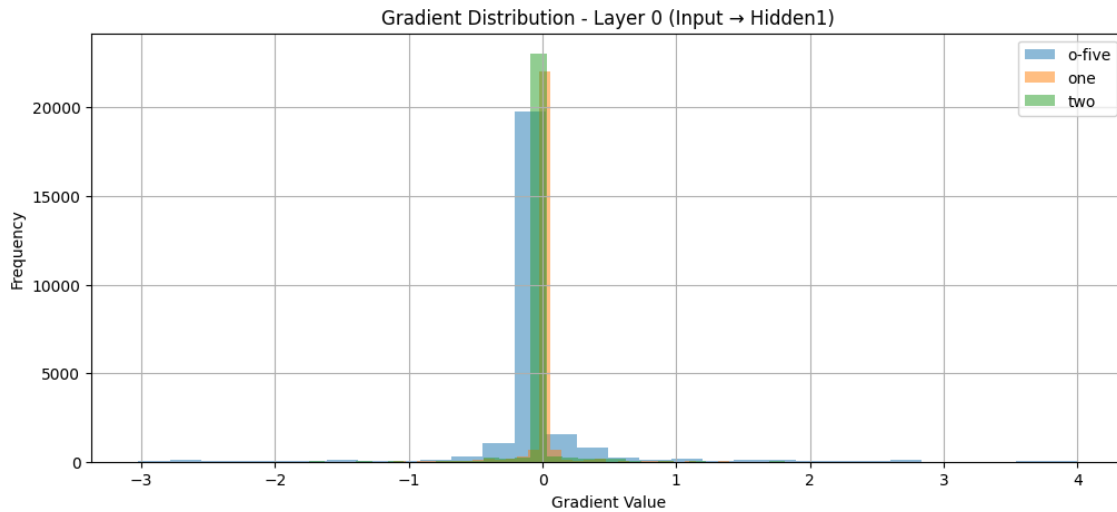
Grafik training loss dan validation loss tiap epoch



Distribusi bobot layer 1



Distribusi gradien bobot layer 1



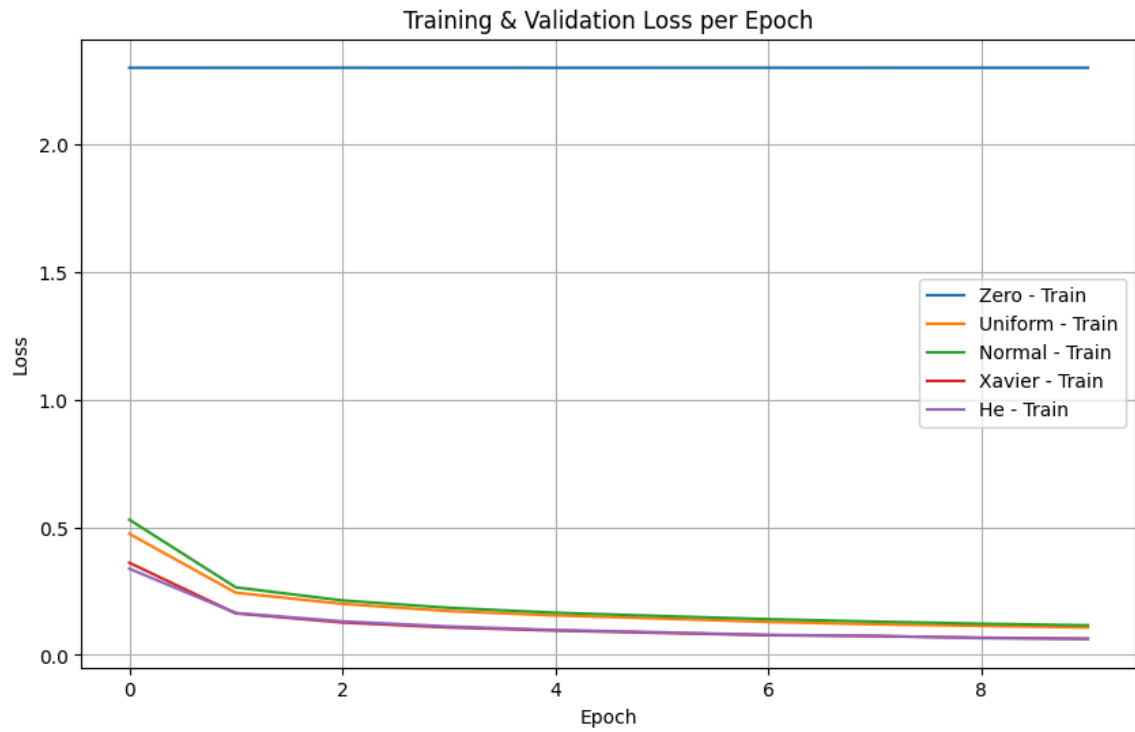
Pengujian menunjukkan bahwa variasi learning rate berdampak pada kinerja model, dengan nilai 0.2 menghasilkan akurasi tertinggi (96.48%), disusul oleh 0.1 (96.37%) dan 0.05 (95.84%). Mungkin learning rate yang sedikit lebih besar menghasilkan performa yang lebih baik karena membantu model mencapai konvergensi lebih cepat dan keluar dari daerah loss yang kurang optimal di awal pelatihan. Hal ini juga terlihat pada grafik loss, di mana learning rate 0.2 menunjukkan penurunan loss yang paling cepat dan stabil sejak awal epoch.

D. Pengaruh Inisialisasi Bobot

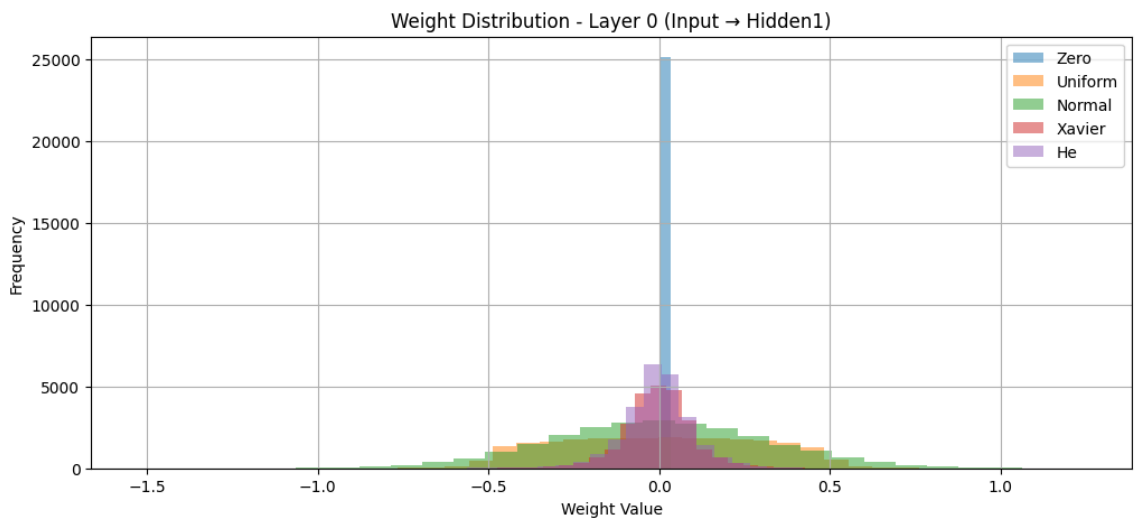
Tabel hasil pengujian terhadap inisialisasi bobot

Inisialisasi Bobot	Akurasi
Zero	11.40%
Uniform	95.59%
Normal	95.61%
Xavier	96.37%
He	96.59%

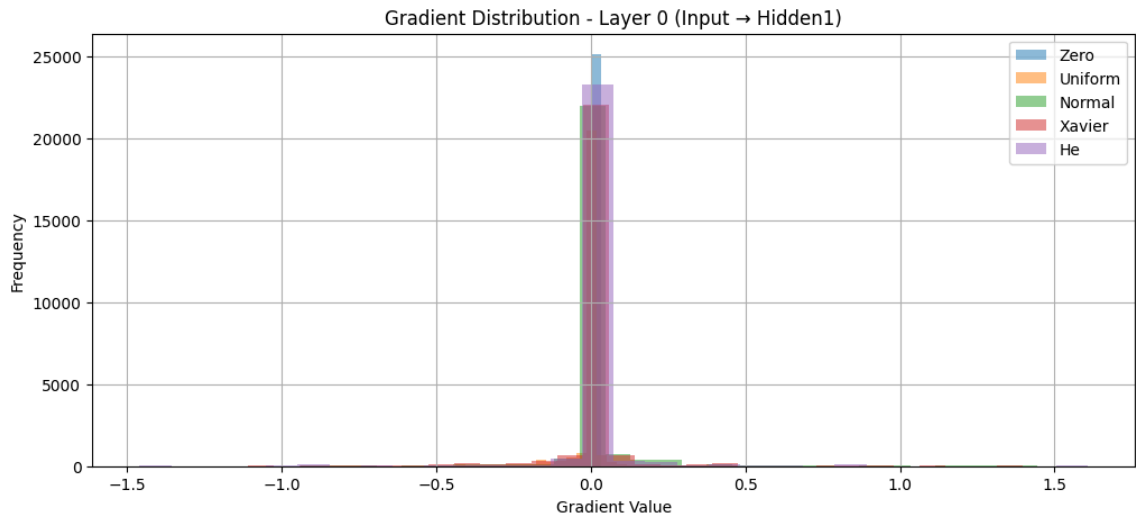
Grafik training loss dan validation loss tiap epoch



Distribusi bobot layer 1



Distribusi gradien bobot layer 1



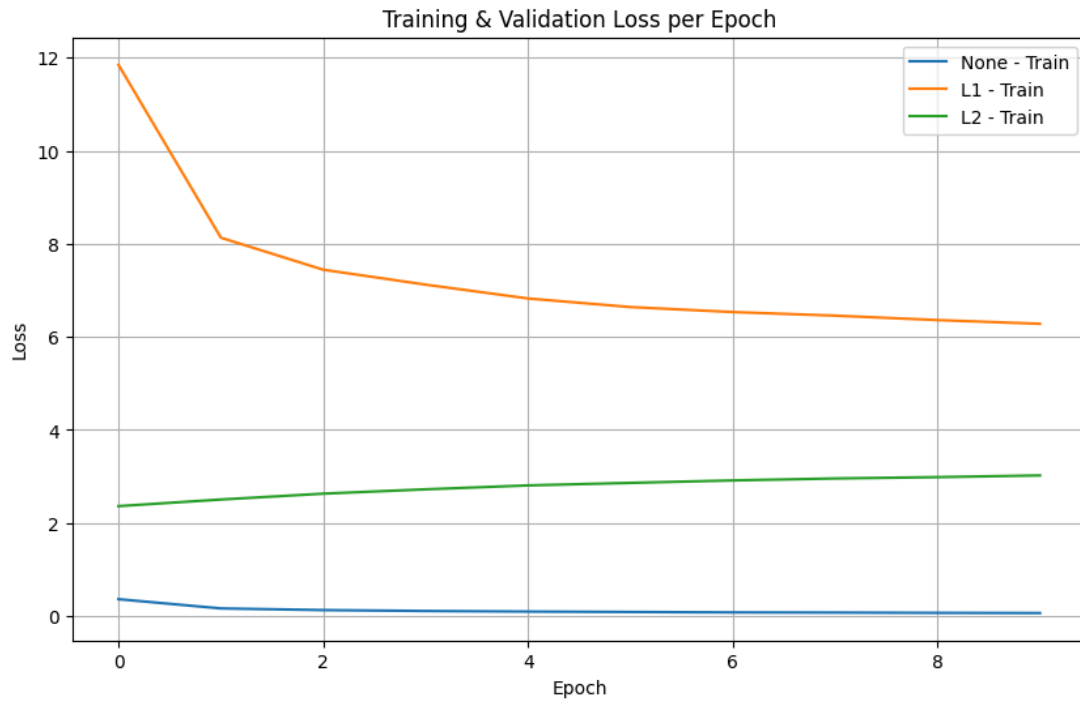
Pengujian menunjukkan bahwa inisialisasi bobot sangat memengaruhi performa model. Inisialisasi Zero menghasilkan akurasi paling rendah (11.40%) dan gagal menurunkan loss selama pelatihan, seperti terlihat pada grafik loss yang stagnan. Hal ini mungkin terjadi karena semua neuron menerima bobot awal yang identik sehingga tidak belajar secara berbeda. Sebaliknya, metode inisialisasi seperti Xavier dan He menghasilkan akurasi tertinggi (96.37% dan 96.59%) dengan penurunan loss yang cepat dan stabil. Dapat dilihat bahwa distribusi bobot awal yang lebih tersebar dan gradien yang tetap seimbang membuat model dapat belajar lebih efektif.

E. Pengaruh Regularisasi

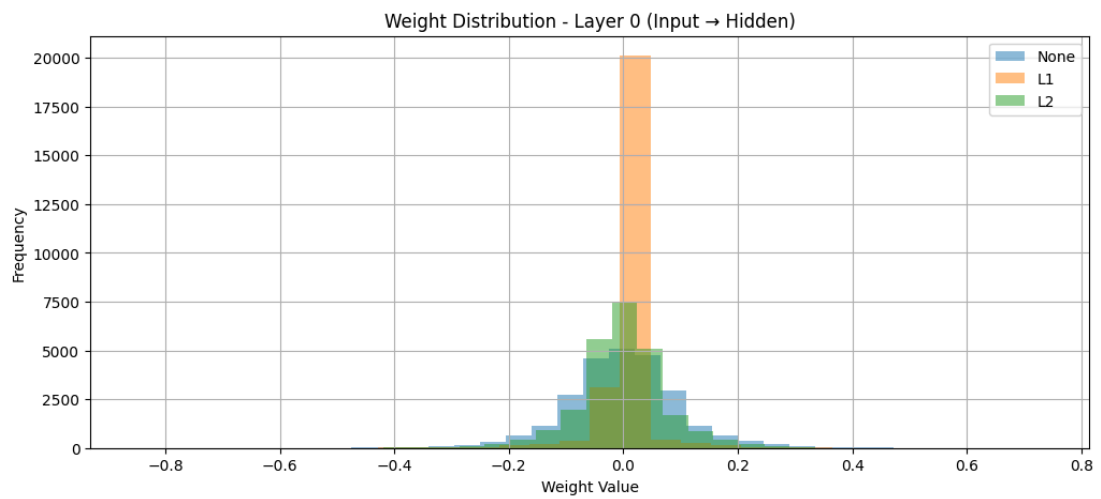
Tabel pengujian terhadap regularisasi

Regularisasi	Akurasi
Tidak ada	91.94%
L1	91.66%
L2	91.94%

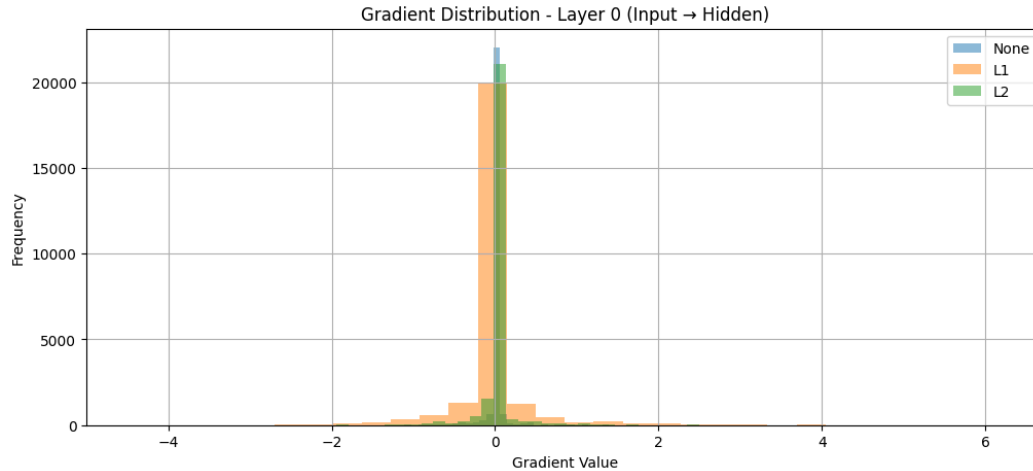
Grafik training loss dan validation loss tiap epoch



Distribusi bobot layer 1



Distribusi gradien bobot layer 1



Hasil pengujian menunjukkan bahwa penggunaan regularisasi belum memberikan perbedaan akurasi yang signifikan pada skenario ini. Baik tanpa regularisasi maupun dengan L2 yang menghasilkan akurasi yang sama (91.94%) dan L1 dengan akurasi yang sedikit lebih rendah (91.66%). Grafik loss menunjukkan bahwa ketiga pendekatan relatif stabil sepanjang pelatihan, meskipun L1 cenderung lebih lambat konvergen. Distribusi bobot pada L1 yang lebih terkonsentrasi mungkin berkontribusi terhadap hal tersebut. Mungkin efek regularisasi akan lebih terlihat pada data yang memiliki lebih banyak noise dan mudah membuat model mengalami overfitting.

F. Perbandingan dengan Library *sklearn*

act_function	configuration	accuracy	
		model	sklearn
reLU	depth: 1, width: 64	93%	96.3%
eLU	depth: 1, width: 64	92%	-
h_tan	depth: 1, width: 64	86.6%	91.8%
sigmoid	depth: 1, width: 64	88.6%	93.9%
linear	depth: 1, width: 64	91.2%	91.2%

Perbandingan dengan *sklearn* menunjukkan bahwa model yang kami dikembangkan memiliki performa yang cukup kompetitif. Akurasi pada fungsi aktivasi ReLU dan linear mendekati hasil *sklearn*, sementara selisih pada fungsi lain kemungkinan disebabkan oleh perbedaan implementasi atau parameter default. Secara keseluruhan, model sudah menunjukkan kinerja yang hampir sebanding dengan library standar.

BAB III

KESIMPULAN DAN SARAN

KESIMPULAN

Pada tugas besar ini, kami telah berhasil mengimplementasikan algoritma Feed Forward Neural Network (FFNN) dari awal tanpa menggunakan library high-level seperti TensorFlow atau PyTorch. Implementasi ini mencakup fungsi aktivasi, metode inisialisasi bobot, berbagai fungsi loss, serta regularisasi L1 dan L2, yang seluruhnya diintegrasikan ke dalam kerangka kerja FFNN modular yang fleksibel dan dapat digunakan untuk eksperimen lebih lanjut.

Melalui serangkaian pengujian, kami mengevaluasi pengaruh dari berbagai aspek terhadap performa model, antara lain: fungsi aktivasi, learning rate, kedalaman dan lebar jaringan, metode inisialisasi bobot, fungsi loss, dan regularisasi. Hasil pengujian menunjukkan bahwa pemilihan fungsi aktivasi ReLU, inisialisasi He atau Xavier, serta penggunaan regularisasi L2 secara konsisten menghasilkan bobot yang stabil dan akurasi yang tinggi.

Selain itu, kami juga membandingkan performa model FFNN yang diimplementasikan sendiri dengan model MLPClassifier dari library sklearn. Hasil perbandingan menunjukkan bahwa model sklearn memiliki waktu pelatihan yang lebih cepat dan performa yang lebih stabil secara default, namun model kami lebih fleksibel dan memberikan insight yang lebih dalam terhadap proses pelatihan, distribusi bobot, dan gradien. Dengan demikian, implementasi dari awal ini lebih cocok untuk keperluan edukasi, eksperimen lanjutan, dan analisis mendalam terhadap komponen-komponen pembelajaran jaringan saraf.

SARAN

Terdapat beberapa saran yang ingin kami sampaikan dalam pengerjaan tugas besar ini, yaitu:

1. Optimasi Visualisasi dan Manajemen Output: Banyaknya visualisasi yang dihasilkan selama eksperimen menyebabkan ukuran file notebook menjadi sangat besar. Oleh karena itu, disarankan untuk menyimpan visualisasi sebagai file eksternal dan menampilkannya hanya saat diperlukan agar notebook tetap ringan dan responsif.
2. Pemisahan Modul Visualisasi dan Training: Agar kode lebih terstruktur dan mudah dikembangkan, disarankan untuk memisahkan fungsi-fungsi visualisasi ke dalam modul tersendiri, terpisah dari komponen training utama.


BAB IV

PEMBAGIAN TUGAS

NIM	Nama	Tugas
13522018	Ibrahim Ihsan Rasyid	Regularisasi, notebook test keseluruhan, laporan
13522028	Panji Sri Kuncara Wisma	Visualisasi, save & load, bonus fungsi aktivasi, fix backpropagation, testing mnist dataset, debug dan fix kode ann, laporan
13522076	Muhammad Syarafi Akmal	Kode base ann (forward propagation and backward propagation), bonus fungsi aktivasi, inisialisasi bobot, notebook test partial, laporan

BAB V

REFERENSI

- Link Github: <https://github.com/SyarafiAkmal/MengajiLur>
- Spesifikasi Tugas:  Spesifikasi Tugas Besar 1 IF3270 Pembelajaran Mesin
- Salindia Kuliah Pembelajaran Mesin FFNN: [Salindia FFNN](#)
- Dataset MNIST: <https://www.openml.org/d/554>